# A Game Engine as a Generic Platform for Real-Time Previz-on-Set in Cinema Visual Effects

Timothée de Goussencourt, Jean Dellac, Pascal Bertolino

# A game engine as a generic platform for real-time previz-on-set in cinema visual effects

Timothée de Goussencourt[1,2] Jean Dellac[2] Pascal Bertolino[1]

[1] GIPSA-lab, Grenoble Alpes University, Grenoble, France
{timothee.de-goussencourt,pascal.bertolino}@gipsa-lab.fr
[2] Solidanim, Paris, France
{tim,jdellac}@solidanim.com

**Abstract.** We present a complete framework designed for film production requiring live (pre) visualization. This framework is based on a famous game engine, *Unity*®. Actually, game engines possess many advantages that can be directly exploited in real-time pre-vizualization, where real and virtual worlds have to be mixed. In the work presented here, all the steps are performed in *Unity*: from acquisition to rendering. To perform real-time compositing that takes into account occlusions that occur between real and virtual elements as well as to manage physical interactions of real characters towards virtual elements, we use a low resolution depth map sensor coupled to a high resolution film camera. The goal of our system is to give the film director's creativity a flexible and powerful tool on stage, long before post-production.

**Keywords:** Depth map, Real-time, Compositing, Virtual Production, Previz on-set, Real-virtual interaction, Unity

## 1 Introduction

Since the beginning of film industry, movie makers like to compose a final scene by taking parts from different sources. With the coming of computer graphics in 90's it became possible to generate virtual graphics (assets) such as virtual backgrounds or virtual characters and to mix them with real images.This action is commonly called compositing. It must deliver a final image in accordance to the storyboard and the film director's wishes. At the moment, this is most of the time a post-production process taking a lot of time and money. Previz on-set is an emerging discipline in film production to achieve live previzualisation. It fills the gap between the storyboard or the full 3D preview sequence (a.k.a. animatik) and the final image (figure 1). The main objectives of previz on-set is to improve film's director creativity during filming and also to speed-up the post-production step.

The solution that we present is a novel and distorted usage of a well-know tool of the game designers, namely the video game engine. With it, the output real-time previz provided to the film's director is built from several inputs: the HD images from the film camera, the virtual assets from 3D libraries, the depth

map of the scene provided by a depth sensor and the interaction between the real world and the virtual assets. The software we chose, that permits this real-time mixing is the powerful and widely-used video game engine *Unity*® [26]. Not only *Unity* is able to handle and merge the above elements in real-time but it also provides a built-in interface that allows an operator to edit in real time the content of the resulting scene without any extra complex coding. However, *Unity* is quite open and it is rather simple to implement specific real-time processing with shaders as it is mandatory in the kind of application we tackle. Apart from a first use of *Unity*® in previz [18] but only for the visualisation of special effects, to our best knowledge our work is the first one to use a game engine in the particular field of previz and more generally in the mix of virtual and real worlds. In [5], we presented a preliminary version of this work. The novelty and originality of the proposed work is that we show that using a game engine is also efficient for the following aspects: (1) managing physical interactions between real and virtual worlds, (2) coupling our previous system with a camera tracking system, (3) re-using all the information captured in real-time in post-production, (4) improving the realism of the virtual assets with the real scene lighting. At last, the results presented here are also more numerous than in [5].

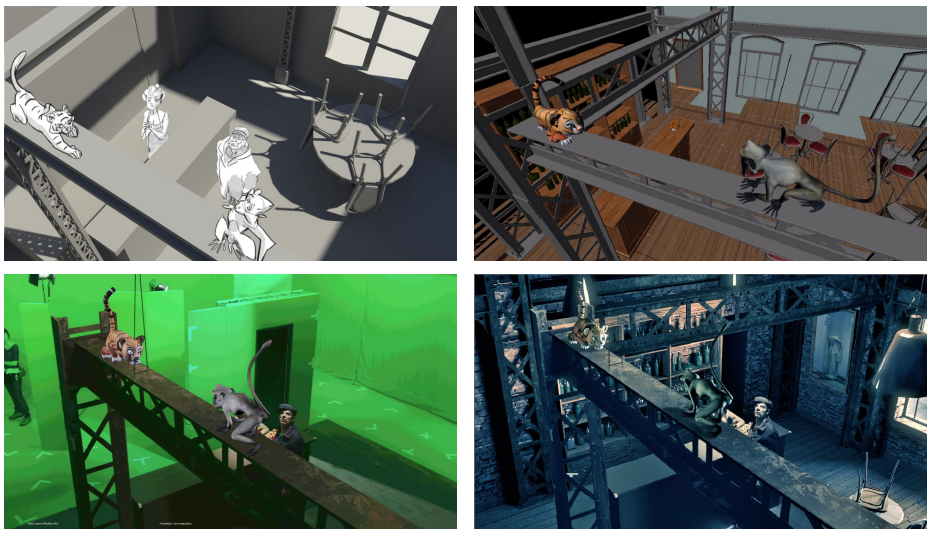

**Fig. 1.** Film building process. From left to right: first row: storyboard and animatik. Second row: previz on-set and post-production final compositing. (*all images from the PREVIZ project*)

This paper is outlined as follows: in section 2 we present some technical background features that are the bases of our solution and we compare them with relative works on this field. Section 3 describes the main points of our

approach and how they are technically implemented into the game engine. The paper will be concluded with some results in section 4.

## 2 Technical background

The depth information of the real scene is mandatory in previz applications and it must provided by a fast hardware system. Among them, the first generation of Kinect is based on an active stereo techniques using structured light [23]. The use of an infrared pattern has the advantage not to disturb the visible spectral domain of the scene. Another technique using infrared is the time of flight (ToF) technology in which the time that it takes for an infrared ray to hit an object of the scene and go back to the sensor is measured, providing the corresponding depth [10]. The second version of the Kinect we are using in this work uses this technology and is a good compromise in terms of price and quality. The system we present is designed for indoor small scale scenes, in a controlled environment which may correspond to an important part of virtual production needs.

Some techniques to mix high definition color frames with depth sensors have already been developed. For instance [3] uses a trifocal rig to generate high precision depth maps: the authors combine a stereo system with a monocular depth sensor, similar to [29,12,25,8]. Then, a complex global optimization workflow is needed to merge data. [21] uses a graph-cut optimization whereas [7] formulates a convex optimization problem to make depth image upsampling. All of these methods use custom rigs. The *Arri* company develops a prototype coupling professional video sensor with a ToF depth sensor [11]. Conversely, the method we propose can deal with every film camera.

Our fusion method is implemented into the popular game engine *Unity* [26]. This platform was chosen because of its great versatility and huge developers community. The graphics engine incorporated into *Unity* is built on top of most modern graphics API, achieving real-time rendering performance, which was an important prerequisite for our project. In our case, the *Unity* software is not used to build a game but to perform and display real-time image processing.

An important step of our method is compositing. Traditional compositing techniques are using a layer scheme, where all objects of the scene are precisely ordered from the closest to the furthest layer. This kind of representation is static and does not permit all the occlusion cases. In our case the final preview image is generated by composing the virtual frame with the real one regarding the depth values of their content. This technique is called depth based compositing. The registered depth maps of virtual and real scenes are compared on the GPU to generate a per pixel matting mask [2]. One of the big challenges of compositing is to ensure seamless integration of virtual layers with the camera footage [14]. In our method we focus particularly on automatic lighting and grading.

In order to add automatic interaction between real world and virtual contents, we also use body segmentation functionalities provided by the Kinect SDK [22]. Some related works dealing with real/virtual mixing can be found in [1,13].

## 3 Exploiting the game engine

This section lists the main contributions of the game engine as a natural and powerful platform for previz on-set 2. We show that the main following steps of previz are successfully addressed through *Unity*: acquiring and registering the data flows, controlling the scene, compositing the scene by mixing real images and virtual workflow, lighting the virtual assets, managing physical interactions between an actor and virtual objects, creating data backup of the whole for post-production.

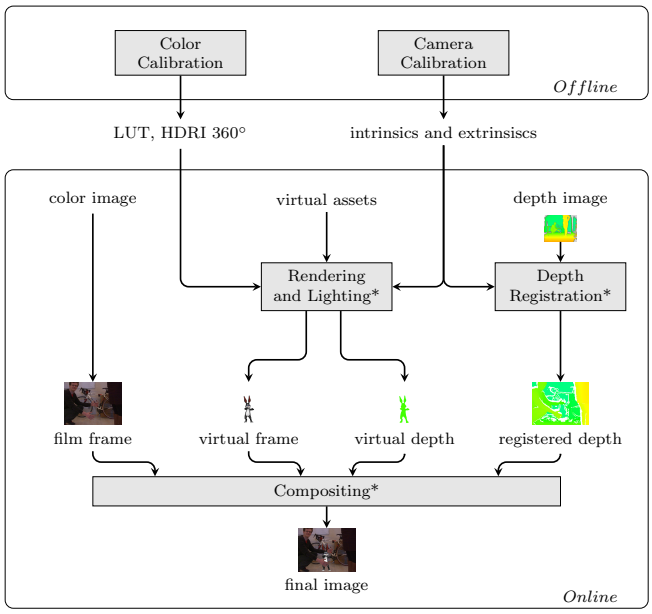


**Fig. 2.** Overall system workflow. Offline process is executed once at the beginning. Online process is executed in real-time for each frame into the game engine. Blocks marked by a * use shaders in their process.

### 3.1 Image acquisition

A key point of our approach is to use the depth of the real scene. To that purpose, a Kinect 2 depth sensor is rigidly fixed to a high resolution film camera as shown in figure 3. The sensors baseline is minimized in order to avoid shadow problems.

The calibration of the two cameras is classical and is performed offline: the intrinsic parameters of each camera are retrieved. Then an extrinsic calibration is done to compute the relative position of the depth sensor to the film camera. A well-known technique for calibrating a camera is based on chessboard pattern

**Fig. 3.** Our system using different film cameras coupled with the Kinect 2. Left: Blackmagic Studio HD with 14mm lens. Right: Panasonic AG-AF100A with a 18mm lens

recognition [27,28]. This calibration is computed only once at the beginning, all calibration parameters are saved, then loaded to the game engine at run time.

The HD color video stream is transferred from the film camera to the computer with an SDI to USB3 acquisition card. The HD color frame (1920 × 1080 pixels) is stored in a 2D texture, a specific structure that the GPU can handle. About the depth, the sensor delivers a 512 × 424 pixels resolution, 16 bits per pixel, real-time depth map. The color sensor available on the Kinect 2 is not used. This buffer is also stored into a 2D texture structure, for a GPU use. As there is no trigger input in the Kinect to synchronize it with an external clock, we set up a circular frame buffer where each depth frame is stored with its corresponding timestamp. When a film frame arrives, the closest (in time) depth frame is selected and associated to the HD color frame. This is a way to compensate the small delay that can occur between depth and color streams.

### 3.2 Depth and color image registration

The acquired HD color images and depth images do not fit at all: the pixel density of the color image is much higher than the one of the depth image: in our context, 1 depth pixel correspond to about 30 color pixels. Furthermore, the field of view of the depth sensor is fix and may largely differ from the one of the HD camera that depends on the used lens. The dense registration of the depth image with the color image is performed in *Unity* with a shader. Shaders are small scripts that are executed by the GPU. In this particular case, the shader is used to back-project the 3D information from the depth sensor into the main camera screen space, for each frame. The current depth texture and a grid mesh with the dimension of the depth map (512 × 424 vertices) are sent to the GPU. Each vertex is displaced in regards to the depth map value and the depth sensor intrinsic parameters. This way a 3D representation of the scene is computed in real-time using GPU capabilities. This 3D mesh is now observed according to the film camera point of view using the calibration settings. Re-projection of the low resolution depth map becomes a dense interpolation, fitting the resolution of the film camera. More details are presented in [5]. Note that shaders may also be used for many kernel-based image filtering.

### 3.3 Editing and controlling

A game engine is commonly used to render in realtime a virtual scene. This is an optimised framework to deal with a lot of 3D assets, lighting and physical interactions. In our context we consider the game engine as a powerful video mixer. Several inputs are mixed together to produce the preview image, using all the great features a game engine can offer. You can program the game engine behaviour by scripting functions placed in a main loop scheme. Each time a new film frame is rendered, this main loop is called and executed.

*Unity* also provides a live editor mode to control in realtime each parameter of each element of the virtual scene, with a graphic user interface (GUI). That means you can modify or tune many things online: lighting parameters, add 3D assets, displace virtual contents, trig animation ... The main control settings are already included in the game engine framework, and it is also easy to add custom controls relative to your scripts into the editor interface. This point is important because the user is thus able to change online the layout of the scene in order to answer the film director's wishes. Is is also possible to build your scene as a standalone executable to achieve better performances. In this case you should construct a GUI control panel to allow online editing of the 3D scene.

### 3.4 Depth based compositing

The final step of our pipeline is the compositing of all the virtual assets with the real scene with the handling of occlusions. The point of view of the virtual scene render is either defined by hand for a static camera shot or possibly triggered by a camera tracking system for a moving shot. This render point of view, defined by a virtual camera, matches the pose and lens configuration of the real film camera. This way the virtual frame perfectly overlaps the film frame.

To handle occlusions we ask the game engine to grab the depth map of the virtual content. Then it is possible to compare the real scene depth map with the virtual scene depth map to generate a binary mask coding whether a pixel of the final rendering comes from the virtual asset frame or from the film camera frame. This mask is smoothed using a blur filter to attenuate hard discontinuities between background and foreground. The final compositing is then computed using the classic *over* operator [2], see Figure 4.

### 3.5 Automatic lighting and grading

In order to blend seamlessly the virtual assets with the real world we need to replicate the lighting [6] and color response [15] of the real scene. To achieve that, a unique high dynamic range image (HDRI) of the scene using a 360° camera (figure 5, top) is captured offline. Then the *Greta MacBeth ColorChecker* is used to calculate two transfer functions $A$ and $B$ stored into look up tables (LUT) (figure 5, bottom). $A$ transfers a neutral colorspace (sRGB) to the colorspace of the film camera. $B$ transfers the colorspace of the 360° camera to a neutral colorspace. The HDRI is transformed to the neutral colorspace using $B$. Then

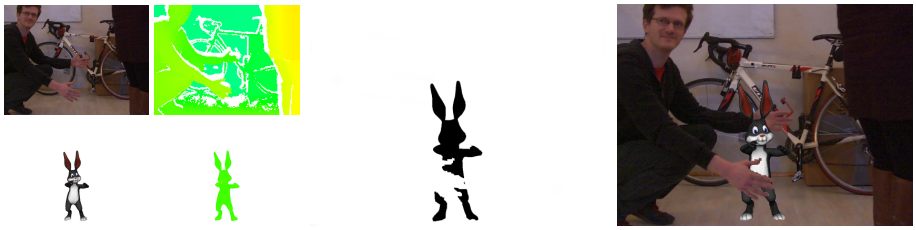

**Fig. 4.** Example of compositing. Left: first row: film frame and registered depth map, second row: virtual character frame and depth map. Middle: binary mask. Right: final compositing

the result is used to light the virtual assets in neutral space using image based lighting (IBL). Finally $A$ is applied to the rendered virtual elements, so they match the color response. *Unity* natively supports a real-time optimised IBL system, again with the use of a shader.

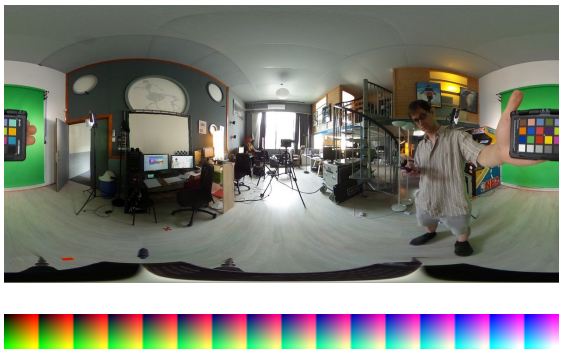

**Fig. 5.** Top: sample of a 360° high dynamic range image used to light the virtual assets. The color checker can be seen on each side of the picture. Bottom: LUT that maps the HDRI colorspace to a neutral colorspace stored into a 2D texture

### 3.6 Interaction between real and virtual elements

Game engines include interesting features like physics engine. That means each virtual element could physically behave like real objects. It is possible to set-up a gravity, a mass for each object and a collision detection. In our case we focus on interaction between real actors and virtual objects. Even if is technically possible to compute collision with the detailed mesh from the depth map, this will affect real-time performances. We solve this problem by using a 3D schematic representation of the human skeleton of the actors. Human pose recognition is computed with a method described in [22] and included in the Kinect SDK.

When a 3D skeleton collides with a virtual object, the game engine trigs an event. This event can be bound to a script where any physical action can be planned. An example of brick wall destruction is showed in figure 6.

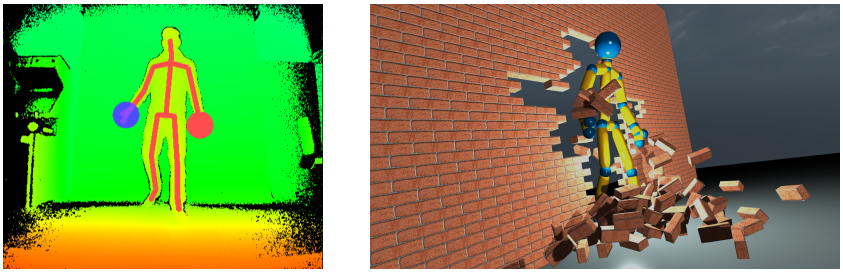

**Fig. 6.** Interaction example. Left: skeleton detection in the depth map. Right: 3D representation of the skeleton in the game engine editor. Collision detection between the skeleton of the real actor and the virtual brick wall

### 3.7 Data backup and post processing

Previz on-set is a precious help during shooting but this is not the end of the film building process. The second main role of previz is to facilitate and improve post-production. To do this, each data stream (depth, camera tracking) is backuped independently during the shooting. This way it is possible to virtually replay the scene, but this time without any realtime constraint. But even is it possible to slow down a game engine to achieve a better rendering or give more time to computation, it is optimised for real time. So, some professional software are dedicated to compositing, such as Nuke [19] or Natron [17], and very popular in post-production studios. This is why we choose to convert the data into an openEXR [20] image sequence. This format was originally developed by Illuminated Light and Magic and it is now a standard in film industry. It supports multi-channels and until 32bits precision per channel. It is also possible to add custom information in headers, like 3D pose of the camera. This way previz data are stored into a well known formalism that professionals are used to work with.

## 4 Experiments

The computer used for our experiment is a laptop with an Intel i7 2.8GHz processor, 16Go RAM and a Nvidia Quadro K4100M. The Unity version 4 and 5 were also used. Our system was tested with two different film cameras used traditionally for broadcast applications (see Figure 3). Their setup includes a $18mm$ lens mounted on a Panasonic AG-AF100A and also a $14mm$ lens mounted on a Blackmagic Studio HD camera.

Our compositing results are presented in figure 7. In this sequence we coupled our method with the camera tracking system SolidTrack [24]. This allows us to move the camera freely, conserving the right virtual point of view to render 3D assets. Examples of interaction between an actor and virtual objects are shown in figures 8 and 9. The depth information is used to compute compositing, while a 3D schematic skeleton trigs in background the collision events. The first example shows a brick wall destruction whereas the second one shows hit weightless objects. Figure 10 shows the effect of color transfer to improve the rendering coherence between virtual and real contents. The overall method described here reaches real-time performance to deliver preview image synchronized with the film camera framerate. Some live video samples captured on the laptop screen are available at `https://sites.google.com/site/pascalbertolino/previz2`.

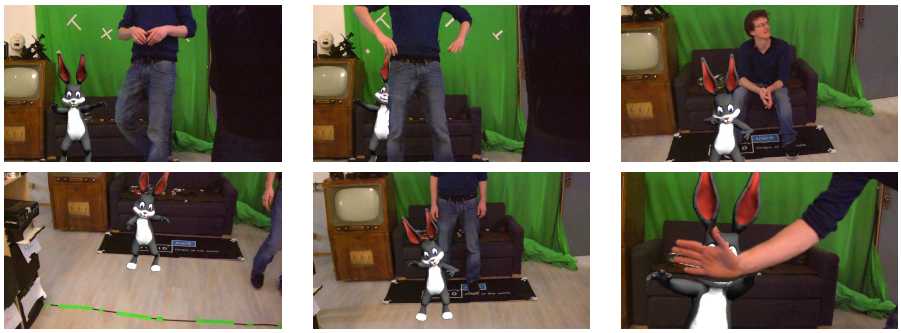

**Fig. 7.** Camera tracking and occlusions. Filmed with the Panasonic AG-AF100A camera with a 18mm lens coupled with the SolidTrack camera tracking system

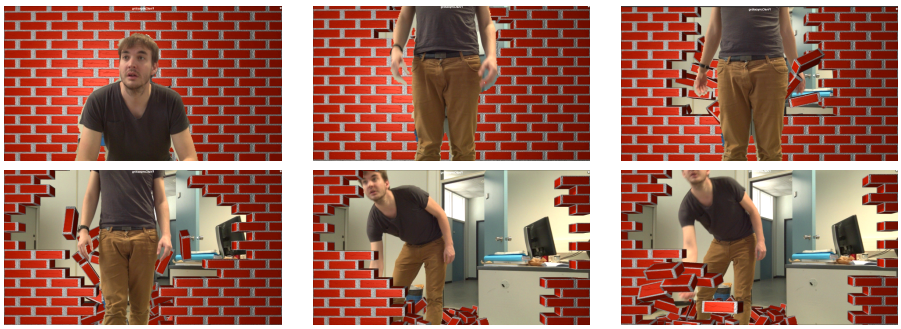

**Fig. 8.** Example of interaction: breaking a virtual wall (from the top left to the bottom right). The character goes back a few seconds, then he hits the wall with his right fist. Filmed with the blackmagic HD studio camera with a 14mm lens

**Fig. 9.** Example of interaction: shooting weightless object (from the top left to the bottom right). The 3D elements are lighted using an HDRI map. Filmed with the blackmagic HD studio camera with a 14mm lens

**Fig. 10.** Automated scene color grading. Left: No color grading is applied. Right: Color grading applied to the virtual scene.

As one can see, the results are not perfect, due to the low quality of the depth map. The depth frame is noisy, and it is quite noticeable at edge. Moreover, the baseline between the film camera and the depth sensor introduces some holes in the registered depth map. But keep in mind that we are looking for previz images: in film industry, the final image is post-processed manually to get the best possible results. However, we plan to perform some guided filtering in order to improve the depth map quality by injecting the film image inside the kernel filter computation. Our method is based on an extension of the joint bilateral filter [16] described in [9].

## 5 Conclusion

In this article we argue why to use a game engine as a platform for previz onset. A game engine offers nice features: optimised for realtime, rendering and

physical engine capabilities. This platform is well adapted on stage and it is adaptive and generic in regards to cameras and specific image processing. The current framework we described here is used in the context of the PREVIZ project [4] dedicated to virtual production, especially previz on-set. All data are recorded online to be re-used in post-production. This way previz on-set is also used as a guide for final compositing. In the future, using this framework, we will propose a database coupling film camera footage with registered depth maps and possibly camera tracking data. This database will be made to better understand how the data produced in previz can be used in post-production.

## References


1. B. Bartczak, I. Schiller, C. Beder, and R. Koch. Integration of a time-of-flight camera into a mixed reality system for handling dynamic scenes, moving viewpoints and occlusions in real-time. In *Proceedings of the 3DPVT Workshop, Atlanta, GA, USA (June 2008)*, 2008.
2. James F Blinn. Compositing. 1. theory. *Computer Graphics and Applications, IEEE*, 14(5):83–87, 1994.
3. G. Boisson, P. Kerbiriou, V. Drazic, O. Bureller, N. Sabater, and A. Schubert. Fusion of kinect depth data with trifocal disparity estimation for near real-time high quality depth maps generation. In *IS&T/SPIE Electronic Imaging*, pages 90110J–90110J. International Society for Optics and Photonics, 2014.
4. G. Briand, F. Bidgolirad, J.F. Zlapka, J.M Lavalou, M. Lanouiller, M. Christie, J. Lvoff, P. Bertolino, and E. Guillou. On-set previsualization for vfx film production. In *International Broadcasting Convention (IBC)*, Amsterdam, Netherland, 2014.
5. T. de Goussencourt and P. Bertolino. Using the unity game engine as a platform for advanced real time cinema image processing. In *ICIP*, Québec city, Canada, September 2015. To appear.
6. Paul E Debevec and Jitendra Malik. Recovering high dynamic range radiance maps from photographs. In *ACM SIGGRAPH 2008 classes*, page 31. ACM, 2008.
7. D. Ferstl, C. Reinbacher, R. Ranftl, M. Rüther, and H. Bischof. Image guided depth upsampling using anisotropic total generalized variation. In *Computer Vision (ICCV), 2013 IEEE International Conference on*, pages 993–1000. IEEE, 2013.
8. V. Gandhi, J. Cech, and R. Horaud. High-resolution depth maps based on tof-stereo fusion. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 4742–4749. IEEE, 2012.
9. Frederic Garcia, Bruno Mirbach, Bjorn Ottersten, Frédéric Grandidier, and Ángel Cuesta. Pixel weighted average strategy for depth sensor data fusion. In *Proceedings - International Conference on Image Processing, ICIP*, pages 2805–2808, 2010.
10. S Burak Gokturk, Hakan Yalcin, and Cyrus Bamji. A time-of-flight depth sensor-system description, issues and solutions. In *Computer Vision and Pattern Recognition Workshop, 2004. CVPRW'04. Conference on*, pages 35–35. IEEE, 2004.
11. Thomas Hach and Johannes Steurer. A novel rgb-z camera for high-quality motion picture applications. In *Proceedings of the 10th European Conference on Visual Media Production*, CVMP '13, pages 4:1–4:10, New York, NY, USA, 2013. ACM.

12. U. Hahne and M. Alexa. Combining time-of-flight depth and stereo images without accurate extrinsic calibration. *International Journal of Intelligent Systems Technologies and Applications*, 5(3):325–333, 2008.
13. S. Izadi, D. Kim, O. Hilliges, D. Molyneaux, R. Newcombe, P. Kohli, J. Shotton, S. Hodges, D. Freeman, A. Davison, et al. Kinectfusion: real-time 3d reconstruction and interaction using a moving depth camera. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*, pages 559–568. ACM, 2011.
14. Georg Klein and David Murray. Compositing for small cameras. In *Proceedings of the 7th IEEE/ACM International Symposium on Mixed and Augmented Reality*, pages 57–60. IEEE Computer Society, 2008.
15. Martin Knecht, Christoph Traxler, Werner Purgathofer, and Michael Wimmer. Adaptive camera-based color mapping for mixed-reality applications. In *Mixed and Augmented Reality (ISMAR), 2011 10th IEEE International Symposium on*, pages 165–168. IEEE, 2011.
16. Johannes Kopf, Michael F. Cohen, Dani Lischinski, and Matt Uyttendaele. Joint bilateral upsampling. *ACM Transactions on Graphics*, 26(3):96, 2007.
17. Natron, INRIA. https://natron.inria.fr/, 2015. [Online; accessed 14-May-2015].
18. Lesley Northam, Joe Istead, and Craig S. Kaplan. A collaborative real time previsualization tool for video games and film. In *SIGGRAPH Posters*, page 121. ACM, 2012.
19. Nuke, the Foundry. https://www.thefoundry.co.uk/products/nuke/, 2015. [Online; accessed 14-May-2015].
20. OpenEXR. http://www.openexr.com/, 2015. [Online; accessed 14-May-2015].
21. S. Patra, B. Bhowmick, S. Banerjee, and P. Kalra. High resolution point cloud generation from kinect and hd cameras using graph cut. In *VISAPP (2)*, pages 311–316, 2012.
22. J. Shotton, T. Sharp, A. Kipman, A. Fitzgibbon, M. Finocchio, A. Blake, M. Cook, and R. Moore. Real-time human pose recognition in parts from single depth images. *Communications of the ACM*, 56(1):116–124, 2013.
23. A. Shpunt and Z. Zalevsky. Three-dimensional sensing using speckle patterns, March 5 2013. US Patent 8,390,821.
24. Solidanim SolidTrack system. http://www.solid-track.com/, 2015. [Online; accessed 14-May-2015].
25. Y. Song, C.A. Glasbey, G.W. van der Heijden, G. Polder, and J.A. Dieleman. Combining stereo and time-of-flight images with application to automatic plant phenotyping. In *Image Analysis*, pages 467–478. Springer, 2011.
26. Unity Game Engine. https://unity3d.com/, 2015. [Online; accessed 14-May-2015].
27. Z. Zhang. Flexible camera calibration by viewing a plane from unknown orientations. In *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, volume 1, pages 666–673. IEEE, 1999.
28. Z. Zhang. A flexible new technique for camera calibration. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 22(11):1330–1334, 2000.
29. J. Zhu, L. Wang, R. Yang, and J. Davis. Fusion of time-of-flight depth and stereo for high accuracy depth maps. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8. IEEE, 2008.